TEAM 5
Project #1
Final Report

Ayesha Ahsan
Steven Layton
Ethan Martin
Marty Thompson

Section 1: Hardware Documentation

The construction of the robot is one of the primary tasks for this project. The sub-team for the construction, called the C-team, consists of Ethan Martin and Marty Thompson. Construction was an ongoing process throughout the entire development of the robot. As issues arose with the performance of the robot, corrections in the initial, conceptual design of the robot needed to be made. This included the addition and removal of some hardware components. This paper will describe the final design of the robot, as it was demonstrated on February 14th, 2003.

The robot consists of three main parts:

- Front Drive Train
- Chassis
- Rear rotating guide wheel

Front Drive Train

Front Wheels and Motors (Figures A, B)

The two front drives wheels are composed of two independent axels, mated to two separate driving motors. This allows for real-time speed control for each of the motor which is regulated in the robot's code. The wheels themselves are attached to the outside of the chassis and consist of 2 inch diameter white rims with rubber tread wrapped around for better traction.

Shaft Encoders (Figures B, C)

Two shaft encoders are placed on the center of the drive train, one mounted on the side of each drive motor. A six-holed wheel is attached to the inside of each drive axel which breaks the beam of the encoder sensor and keeps track of the respective revolutions of each axel. If one shaft encoder is sensed to be spinning faster than the other, a routine is present to compensate for this discrepancy.

Gearing (Figure B)

Due to the LEGO motors having tremendously low torque output except at very high output speeds, a high gear ratio was necessary. The smallest LEGO gear containing eight teeth is attached to each drive motor. The largest LEGO gear containing forty teeth is attached to each independent front drive shaft. These gears mesh together creating a ratio of 9:1. This ratio increases the torque output of the motors significantly allowing for more precise movements.

Light Sensors (Figure B)

The light sensors primary uses in this design are to detect changed in the color of the floor, which in this particular project would be black electrical tape. Once the robot "sees" the next black box, it must turn and realign it self towards the next box. Alignment is needs because as the robot proceeds along its path, it may not necessarily head in a perfectly straight course. The light sensors gather data so that these corrections can be made. After making a left turn in the box, the robot then proceeds slowly towards the black line and makes minor adjustments using data from the sensors to ensure it is lined up nicely with the next destination. The light sensors were attached to several LEGO pieces placed in front of their respective drive wheel. The further the distance between the light sensors, the better alignment the robot is able to achieve.

Chassis (Figures A, B, D)

It is very important that the drive motors stay securely in place during operation. Therefore it was necessary to create a brace to ensure the motors stayed properly aligned. Several four inch LEGO strips were placed on top of the two motors, attaching them together, ensuring their stability. The front motors themselves are placed directly on the chassis and immediately in front of the HandyBoard unit. The HandyBoard unit is quite big and bulky, so it is vital that it does not shift around or fall off the robot while in operation. A box-type containment structure was built to brace the HandyBoard. This is composed of two 4.5 inch black LEGO pieces along the sides of the unit with a three inch support in the back. The motor responsible for controlling the rear rotating guide wheel is placed on a platform supported by long LEGO pieces that are firmly attached to the underside of the chassis, directly beneath the HandyBoard.

Rear Rotating Guide Wheel (Figures A, D)

It is necessary to use a rear rotating guide wheel in order to ensure that the robot travels in a straight path. The guide wheel consists of two small ¾ inch diameter wheels attached to an axel that is further attached perpendicularly to another axel. This axel runs straight up through the previously mentioned supporting platform of the rear motor. A beveled gear system is used to allow a vertically mounted motor to spin the guide wheels in a horizontal arc. The main purpose of the guide wheels is to aid the front motors in making a smooth ninety degree turn and to ensure straight forward movement. While the robot is moving straight, the rear motor is applying force used in keeping the guide wheels straight. A LEGO piece is placed in an exact position next to the guide wheels to make certain that they do not rotate beyond zero degrees. Before turning, the rear motor rotates the guide wheels ninety degrees to reduce friction and assist in steering around the turn. Once again, a LEGO piece is attached to the chassis to keep the wheels from rotating greater than ninety degrees.

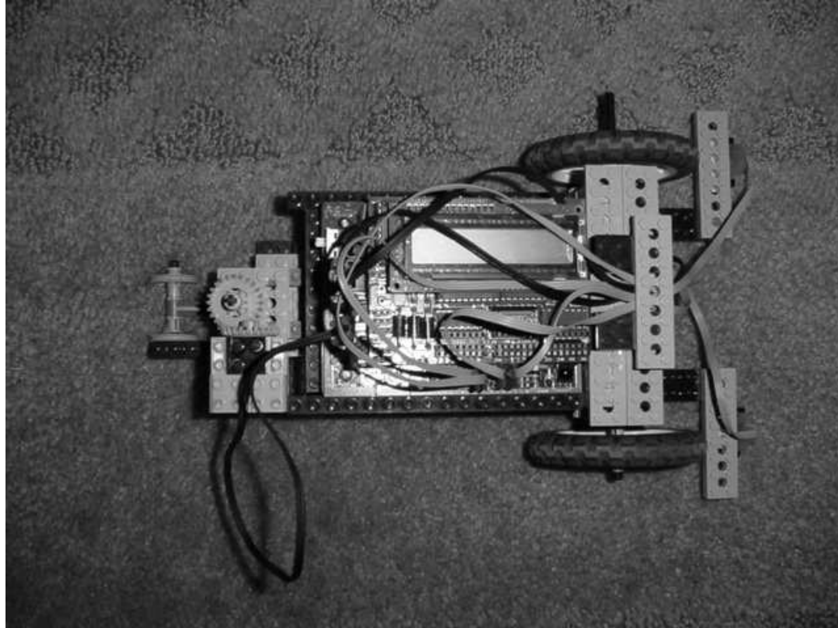


Figure A - (Above) Top view

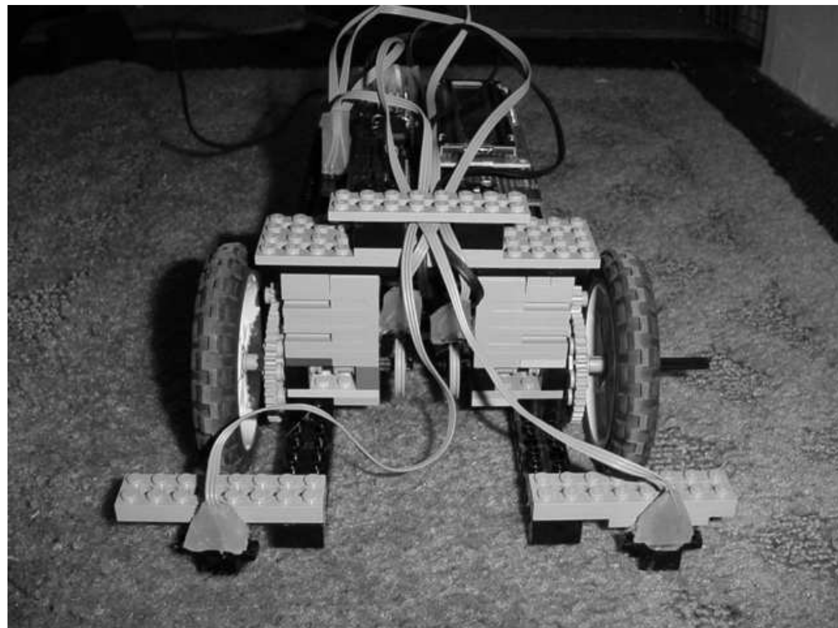


Figure B - (Above) Front view where gears and light sensors are visible

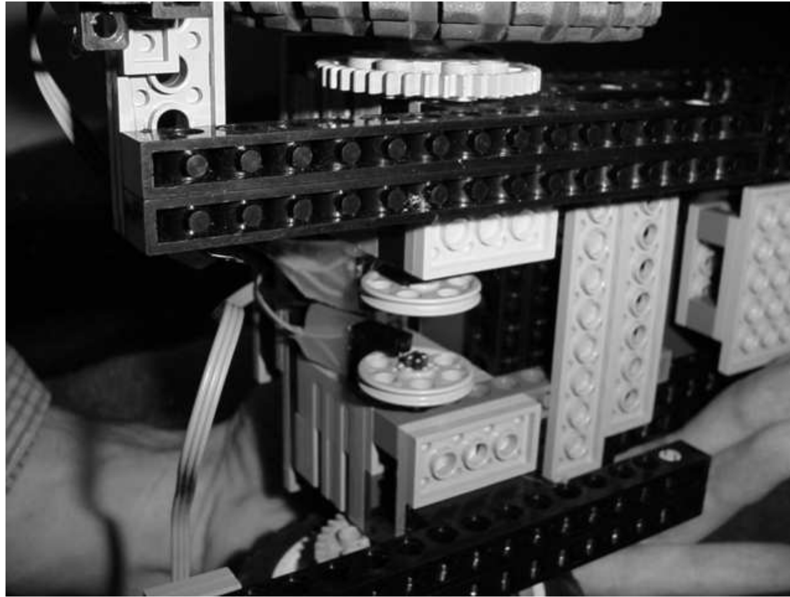


Figure C - (Above) Bottom view where gears and shaft encoders are visible

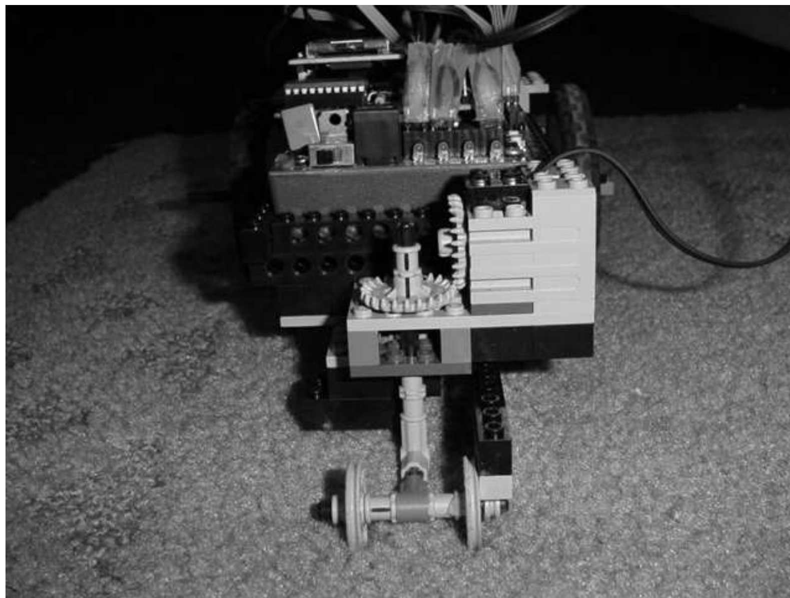


Figure D - (Above) Back view where motorized guide wheel is visible

Section 2: Code Documentation

Introduction

The software design for the robot was one of the primary tasks for this project. The sub-team for the code side of the project, called the C-team, consisted of Ayesha Ahsan and Steven Layton. Software design was an ongoing process throughout the entire project. As the design of robot changed, so did the code. Although no data structures were used, there were many error correction algorithms that were implemented. The code enabled the robot to take environmental readings from light sensors and shaft encoders and using that data to made appropriate decisions that would lead the robot accomplishing its task (i.e. finding the next box).

The following algorithms ensured the success of the robot and will be noted and described in detail in the following section:

- Go straight
- Turn left
- Get lined up

Go Straight

An important part of the robot's success was dependent on its ability to travel from box to box in a straight line. The go straight algorithm ensured this success. The algorithm is as follows: Turn the guide wheels straight. While the total distance traveled is less than a maximum value and while the black tape hasn't been seen, give power to the motors and adjust the heading if necessary.

N.B. The adjust heading algorithm was an error correcting algorithm that compared the right and left encoder readings and decided if motor speed should be reduced. If the distance traveled by the left wheel was greater than the distance traveled by the right, the power of the left motor was reduced by twenty percent (and vice versa). This algorithm was taken from Dr. Miller's code in the \tests directory from the file "dmiller-super-demo.ic" that was included in the IC program.

Turn Left

This algorithm used the guide wheel to make a 90 degree left turn. To ensure that the robot remained unaffected by the force of the turning guide wheel, the right and left wheels where given ample power in the appropriate direction. After the right wheel travels a certain distance (measured by the shaft encoders) the wheels are stopped and the guide wheel is turned back to 0 degrees.

Get Lined Up

Another way to guarantee the success of the robot was to ensure that its initial position after each turn was perpendicular with the box. This was accomplished by using the following algorithm in order of listing, each of which will be discussed briefly:

- **Set the guide wheels straight** - The guide wheels were set straight so the robot would back up straight giving it a better chance of finding a black line.

- **Backup up a short distance** - Backing up proved to be useful after the turning left algorithm just in case the robot had gotten off course and the box was missed.

- **Check the light sensors for black tape** – This check for the black tape by the light sensors was to see if the robot had gotten “lucky” and was correctly aligned. If black tape is not seen, the robot will begin to “wiggle” in the right direction.

- **Wiggle in the appropriate direction until both light sensors see black tape** – Wiggling is performed by setting the motors at opposing speeds in the appropriate direction (if the left sensor sees black tape, the left motor is given power in the reverse direction and the right motor is given power in the positive direction, and vice-versa). This was done until black tape was seen by both sensors or until the maximum number of wiggles allowed was reached. The wiggle algorithm was taken from Dr. Miller's code in the \tests directory from the file “dmiller-super-demo.ic” that was included in the IC program.

  N.B. During the testing phase, the robot ran into complications when only one light sensor found black tape and would wiggle indefinitely or until it was improperly aligned with black tape. Due to this problem, the number of wiggles was limited to ten. This proved to be helpful in helping the robot achieve the next box despite it being off course.

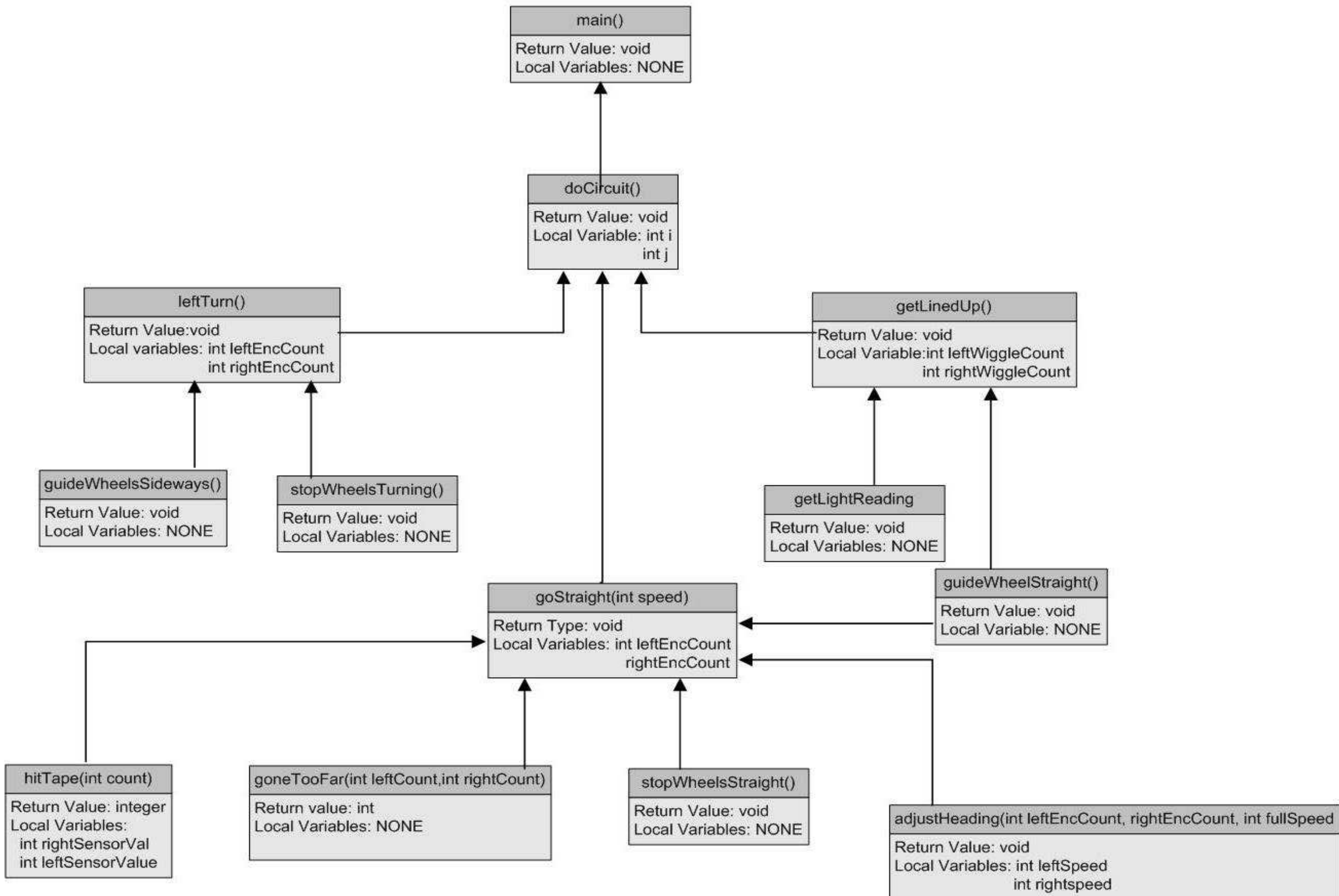
- **Set the guide wheels straight** – Setting the guide wheels straight accounts for the effects of wiggling on the guide wheels, so that the robot is guaranteed to travel straight.

- **Continue on to the next box** - Code was added to reset the encoders if two black lines were seen too close together. This error correction code accounted for when the robot would align itself on the outer black tape and would pick up the inside black tape as the next box.

N.B. During the testing phase, the robot ran into complications when only one light sensor found black tape and would wiggle indefinitely or until it was improperly aligned with the box. Due to this problem, the number of wiggles was limited to ten. This proved to be helpful in helping the robot achieve the next box despite it being off course.

A functionality diagram and source code is attached.

Functionality Diagram



```
/* Team 5
Project #1
Intell. Robotics

This code allows a robot to travel around a circut of four boxes three times.
The adjust heading algorithm and the wiggle algorithm, were both taken from
Dr. Miller's code in the \tests directory from the dmiller-super-demo.ic file.
Changes to these algorithms are noted in the function descriptions */

#define R_MOTOR 0 /* motor port 0 */
#define L_MOTOR 3 /* motor port 3 */
#define R_SENSOR 4 //port where right sensor is
#define L_SENSOR 5 //port where left sensor is
#define RIGHT_ENC 0 //port where right encoder is
#define LEFT_ENC 1 //port where left encoder is
#define GUIDE_MOTOR 1 //guide wheel motor port
#define BLACK_TAPE_VALUE 160 //light reading reading for black tape
#define MIN_TICKS 75 //min ticks before start looking for black tape
#define MAX_TICKS 90 //when you get this far, you have probably missed a box
#define TURN_TICKS 11  /* encoder tics for right angle turn */
int leftLightReading, rightLightReading, numLaps;


//returns TRUE when the robot needs to stop
//uses encoder clicks to make sure the robot does not miss a box
int goneTooFar(int leftCount, int rightCount)
{
   return ((leftCount >= MAX_TICKS) || (rightCount >= MAX_TICKS));
   //beep();
}

//reads the light sensors and stores the values in global vars
void getLightReading()
{
   leftLightReading = analog(L_SENSOR);
   rightLightReading = analog(R_SENSOR);
}

//returns TRUE when robot has hit the tape
int hitTape(int count)
{
   int leftSensorVal, rightSensorVal;

   leftSensorVal = analog(L_SENSOR);
   rightSensorVal = analog(R_SENSOR);

   //unless the encoders have 75 clicks, ignore the tape
   if (count > 75)
     {
       return ((leftSensorVal >= BLACK_TAPE_VALUE) || (rightSensorVal >=
BLACK_TAPE_VALUE));
```

```
  }
  else
    //if you see black tape before 50 clicks, reset the encoders
    if (count < 50)
      {
        if ((leftSensorVal >= BLACK_TAPE_VALUE) || (rightSensorVal >=
BLACK_TAPE_VALUE))
          {
            reset_encoder(LEFT_ENC);
            reset_encoder(RIGHT_ENC);
            return 0;
          }
      }
    else
      return 0;
}

//set the back guidewheels to be straight
void guideWheelsStraight()
{
  motor(R_MOTOR, -10);
  motor(L_MOTOR, 4);
  motor(GUIDE_MOTOR, -200);
  sleep(0.3);
  motor(GUIDE_MOTOR, -20);
  off(R_MOTOR);
  off(L_MOTOR);
}

//set the guidewheels to be sideways
void guideWheelsSideways()
{
  motor(R_MOTOR, -5);
  motor(L_MOTOR, -5);

  motor(GUIDE_MOTOR, 150);
  sleep(0.5);
  motor(GUIDE_MOTOR, 20);
}

//make sure the robot goes straight
//this code was taken from Dr. Miller's in the \tests directory from the dmiller-super-demo.ic file
//local variable names were changed, as well as the function name (from go_straight)
void adjustHeading(int leftEncCount, int rightEncCount, int fullSpeed)
{
  int leftSpeed, rightSpeed;

  leftSpeed = rightSpeed = fullSpeed;

  if (leftEncCount > rightEncCount)
    leftSpeed = (8 * fullSpeed) / 10;
```

```
    else
      if (rightEncCount > leftEncCount)
        rightSpeed = (8 * fullSpeed) / 10;

      motor(R_MOTOR, rightSpeed);
    motor(L_MOTOR, leftSpeed);
}

/* reverses motors briefly for a quick stop */
//code was taken from Dr. Miller's code in the \tests directory from the dmiller-super-demo.ic file
//the function name was changed (from stop_wheels to stopWheelsStraight)
void stopWheelsStraight()
{
    motor(L_MOTOR,-20);
    motor(R_MOTOR,-20);
    sleep(0.1);
    ao();
}

//drives the robot straight and stops in the next box
void goStraight(int speed)
{
    int leftEncCount, rightEncCount;

    reset_encoder(RIGHT_ENC);
    reset_encoder(LEFT_ENC);

    rightEncCount = read_encoder(RIGHT_ENC);
    leftEncCount = read_encoder(LEFT_ENC);

    //get the back wheels straight
    guideWheelsStraight();

    //get the motors going
    motor(R_MOTOR, speed);
    motor(L_MOTOR, speed);

    //if we havent gone too far or hit tape then....
    while (!goneTooFar(leftEncCount, rightEncCount) && (!hitTape(leftEncCount)))
      {
        sleep(0.06);
        leftEncCount = read_encoder(LEFT_ENC);
        rightEncCount = read_encoder(RIGHT_ENC);
        adjustHeading(leftEncCount, rightEncCount, speed);
    }
    //after we stop, beep
    beep();

    //here is where we make sure it stops in the same spot every time
    reset_encoder(RIGHT_ENC);
    reset_encoder(LEFT_ENC);
```

```
    rightEncCount = read_encoder(RIGHT_ENC);
    leftEncCount = read_encoder(LEFT_ENC);
    //go ahead 3 clicks after we need to stop
    while (rightEncCount <= 3)
      {
        rightEncCount = read_encoder(RIGHT_ENC);
        sleep(0.06);
    }
    //stop the robot
    stopWheelsStraight();
}

//stops the wheels when we are turning
void stopWheelsTurning()
{
    motor(R_MOTOR, -10);
    motor(L_MOTOR, 5);
    sleep(0.1);
    ao();
}

//this is the code to align with the black tape
void getLinedUp()
{
    int leftWiggleCount, rightWiggleCount;

    guideWheelsStraight();

    //backup for a bit to give us room....and make sure we dont miss
    motor(R_MOTOR, -15);
    motor(L_MOTOR, -15);
    sleep(1.5);
    getLightReading();
    leftWiggleCount = rightWiggleCount = 0;
    //while both the sensors do not see black....do this
    while ((leftLightReading < BLACK_TAPE_VALUE) || (rightLightReading <
BLACK_TAPE_VALUE))
      {
        printf("left %d right%d\n",leftLightReading, rightLightReading);
        //if the right sensor sees black and the left doesnt...wiggle right
        if (rightLightReading >= BLACK_TAPE_VALUE)
          {
            stopWheelsStraight();
            motor(R_MOTOR, -10);
            motor(L_MOTOR, 20);
            sleep(0.1);
            rightWiggleCount++;
            leftWiggleCount = 0;
            printf("right sees tape\n");
        }
```

```
        else
          //if the left sees black and the right one doesnt....wiggle left
          if (leftLightReading >= BLACK_TAPE_VALUE)
            {
              stopWheelsStraight();
              motor(L_MOTOR, -10);
              motor(R_MOTOR, 10);
              leftWiggleCount++;
              rightWiggleCount = 0;
              sleep(0.1);
              printf("left sees tape\n");
            }
          else
            //if neither one sees black, go forward slowly
            {
              motor(R_MOTOR, 8);
              motor(L_MOTOR, 8);
            }
        //if you wiggle right or left 10 consecutive times, then give up and go straight
        if ((rightWiggleCount >= 10) || (leftWiggleCount >= 10))
          return;

        sleep(0.06);
        getLightReading();
    }
    //    stopWheelsStraight();
}

//do a 90 degree left turn
void leftTurn()
{
    int leftEncCount, rightEncCount;

    sleep(1.0);
    reset_encoder(RIGHT_ENC);
    reset_encoder(LEFT_ENC);

    rightEncCount = read_encoder(RIGHT_ENC);
    leftEncCount = read_encoder(LEFT_ENC);

    //set the back wheels sideways to allow for easy turning
    guideWheelsSideways();

    //get the motors going
    motor(R_MOTOR, 25);
    motor(L_MOTOR, -10);

    //while the right wheel has not gone far enough.....
    while(rightEncCount < TURN_TICKS)
      {
        rightEncCount = read_encoder(RIGHT_ENC);
```

```
        //L_Enc = read_encoder(1);

        printf("Right=%d Left=%d\n", rightEncCount, leftEncCount);
    }
    //stop the wheels
    stopWheelsTurning();
}

//going around the circuit 3 times
void doCircuit()
{
    int i, j;
    numLaps = 0;
    for (i = 0; i < 3; i++)
      {
        numLaps++;
        for (j = 0; j < 4; j++)
          {
            goStraight(25);

            //if we are in the last square, we dont care about aligning ourselves
            if ((j == 3) && (i == 2))
              {}else
              {
                leftTurn();
                getLinedUp();
              }

          }
    }
    ao(); //turn the motors off so we can stop
}


void main()
{
    //wait for the start button
    while(!start_button())
      {
    }

    //enable the encoders
    enable_encoder(RIGHT_ENC);
    enable_encoder(LEFT_ENC);

    //do the circuit
    doCircuit();

    //turn off the encoders
    disable_encoder(RIGHT_ENC);
    disable_encoder(LEFT_ENC);}
```

Section 3: Team Organization and Future Plans

Introduction

For a team to evolve in a project oriented work environment, the team must analyze its internal structure and task allocation methods to determine if the organization and allocation is efficient and sufficient in solving problems. We have selected the following points on team organization and future plans to explain:

- Division of team into sub-teams
- Communication of sub-teams
- Testing phase
- Presentation

Division of Team into Sub-teams

The division of the team into two sub-teams (hardware and software) was beneficial for task allocation. We found however that it was better to unite our efforts in the beginning than at the end of the project. Everyone contributed ideas which were incorporated into the robot, which helped ensure that the conceptual design was correct. When one sub-team hit a hurdle, members of the other sub-team contributed to the brainstorming process until a solution was found. Each sub-team will now be addressed.

- *C-team (Code team)*
  The C-team must account for all possible problems that may arise in the testing phase (hence the need for error correction code). The quicker these problems are noticed and dealt with, the more likely the success of the team. Time is the primary issue here, and the C-team should increase their efforts to meet these demands.

- *H-team (Hardware team)*
  The H-team must realize if the conceptual design is feasible in the early stages of the project or much time is wasted. For the C-team to have ample time for testing, the H-team must provide them with a working prototype that has the ability of accomplishing the desired objectives. The earlier this design is realized, the higher chances of having a successful project.

For future projects, we want to stress the need for dynamic sub-teams to aid in progress when solutions are not easily realized. Also members of sub-teams will be switched, so project members can gain experience on each sub-team. We feel these changes in the sub-team structure will aid in the performance of each sub-team.

Communication of Sub-teams

Integrating the sub-teams (construction and code) is difficult unless communication is ongoing. Thus, communication between the sub-teams was highly stressed and frequent short meetings were beneficial in keeping all members informed of progress. In the remaining projects, frequent team meetings will be implemented by the team.

Testing Phase

The testing phase of the project played more of a role than initially expected. We found that conceptual designs were much easier to come by than were working prototypes. Some prototypes were inherently flawed by their conceptual design, which went unnoticed until the testing phase (the initial design of our robot had a static guide wheel, which turned out to be poor when turning). The need for error correcting algorithms would not have been realized without a strenuous testing phase. So for future projects, we want to be rigorous and thorough in testing the robot. Thus we will extend the testing phase for future projects, so a higher number of possible scenarios can be considered.

Presentation

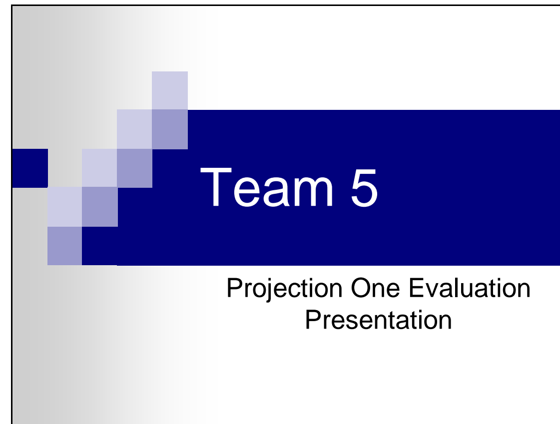
For a fair distribution of tasks, Steven and Marty will present the next project to the class, since Ethan and Ayesha are presenting project 1.

Summary

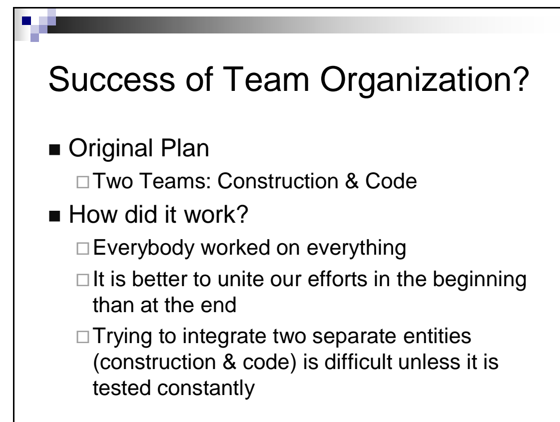
The overall organization of the team was successful and efficient; however the following adjustments should be made: dynamic sub-teams should be allowed so contributions can be made when hurdles are reached, meetings should be held frequently, and the testing phase should be extended. Also, sub-teams will be changed so everyone can gain experience.

Section 4: Presentation

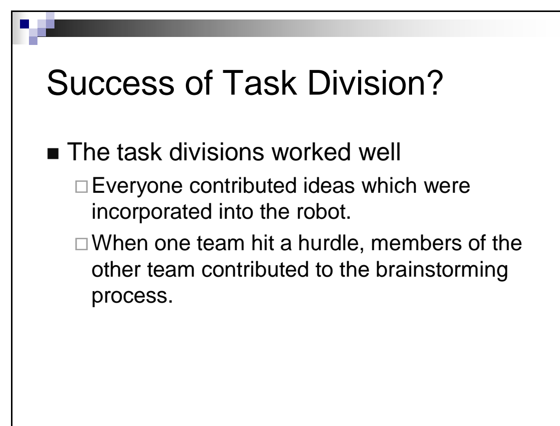
Slide 1



Slide 2



Slide 3

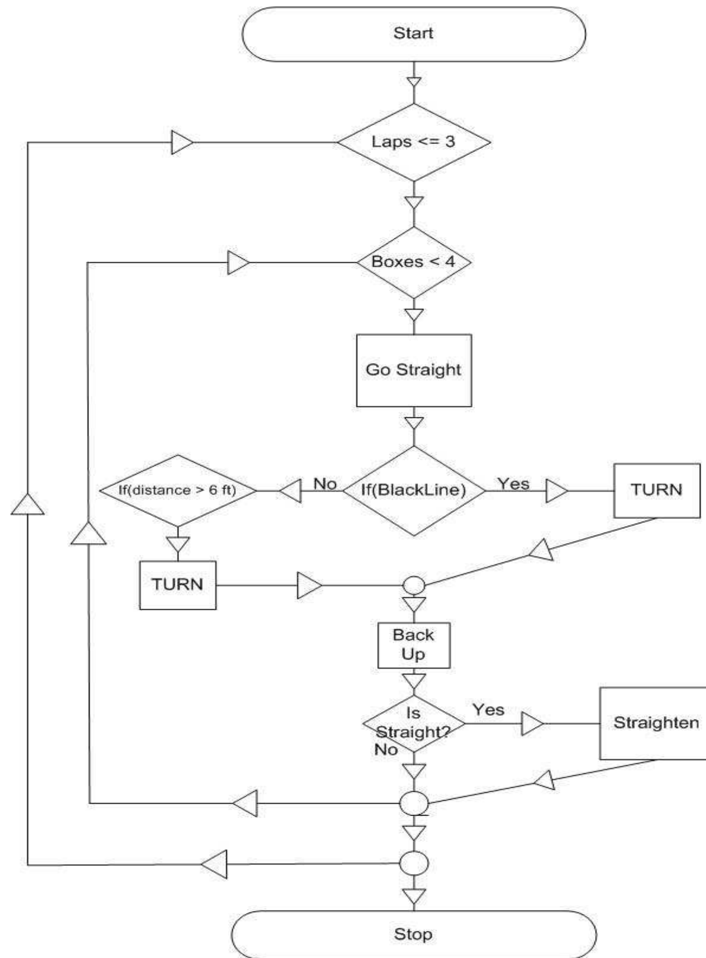


Slide 4

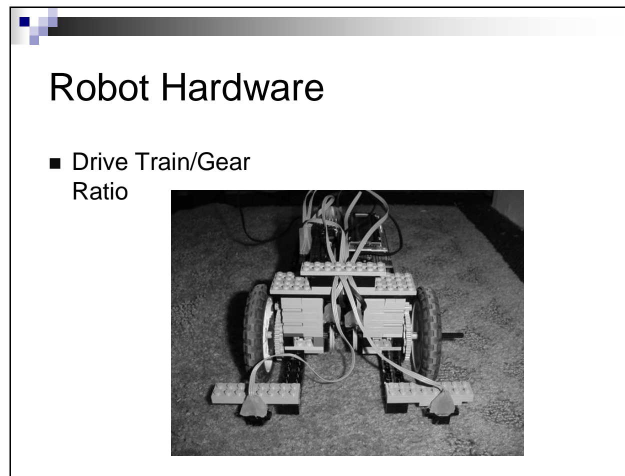
Robot Code

- Utilizes
  - Shaft Encoders
  - Light Sensors
- Error Correction
  - Double black line problem
  - Missing a box completely
  - Alignment (wiggle algorithm)
    - 15 wiggle maximum

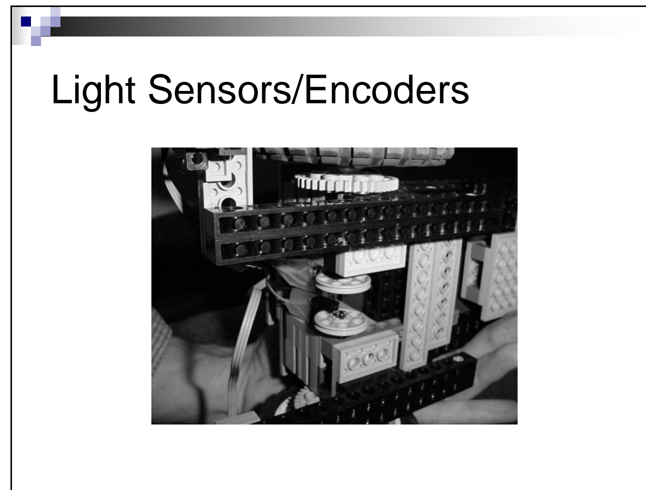
Slide 5



Slide 6



Slide 7



Slide 8